Fundamentals of Computing: Lecture 12

Piyush P Kurur
Office no: 224
Dept. of Comp. Sci. and Engg.
IIT Kanpur

August 24, 2009

Summary of the last class

```
while( C )
{ /* invariant is a condition that is true here */
  S
}
```

How to design loop?

- Write down the desired outcome φ.

Summary of the last class

```
while( C )
{ /* invariant is a condition that is true here */
  S
}
```

How to design loop?

- Write down the desired outcome φ.
- Choose a loop invariant I and a loop condition C such that $I \wedge \neg C \Rightarrow \varphi$.

Summary of the last class

```
while( C )
{ /* invariant is a condition that is true here */
  S
}
```

How to design loop?

- Write down the desired outcome φ.
- Choose a loop invariant I and a loop condition C such that $I \wedge \neg C \Rightarrow \varphi$.
- I is obtained by parameterisation (i.e. replacing constants of φ by variables)

Summary of the last class

```
while( C )
{ /* invariant is a condition that is true here */
  S
}
```

How to design loop?

- Write down the desired outcome φ.
- Choose a loop invariant I and a loop condition C such that $I \wedge \neg C \Rightarrow \varphi$.
- I is obtained by parameterisation (i.e. replacing constants of φ by variables)
- The condition C is the negation of parameterisation.

Summary of the last class

```
while( C )
{ /* invariant is a condition that is true here */
  S
}
```

How to design loop?

- Write down the desired outcome φ.
- Choose a loop invariant I and a loop condition C such that $I \wedge \neg C \Rightarrow \varphi$.
- I is obtained by parameterisation (i.e. replacing constants of φ by variables)
- The condition C is the negation of parameterisation.
- Initialise variables so that I is true in the base case

Summary of the last class

```
while( C )
{ /* invariant is a condition that is true here */
  S
}
```

How to design loop?

- Write down the desired outcome φ.
- Choose a loop invariant I and a loop condition C such that $I \wedge \neg C \Rightarrow \varphi$.
- I is obtained by parameterisation (i.e. replacing constants of φ by variables)
- The condition C is the negation of parameterisation.
- Initialise variables so that I is true in the base case
- Design the body S to preserve the validity of I.

Finding the smallest in a sequence of n numbers

The desired condition is

$$\varphi \equiv s = \min\{a[0], \ldots, a[n-1]\} \text{ (n is a constant here).}$$

Finding the smallest in a sequence of *n* numbers

The desired condition is

$$\varphi \equiv s = \min\{a[0], \ldots, a[n-1]\} \text{ (n is a constant here).}$$

Parameterising we get the invariant

$$I_i \equiv s = \min\{a[0], \ldots, a[i-1]\}.$$

Finding the smallest in a sequence of n numbers

The desired condition is

$$\varphi \equiv s = \min\{a[0], \ldots, a[n-1]\} \text{ (n is a constant here).}$$

Parameterising we get the invariant

$$I_i \equiv s = \min\{a[0], \ldots, a[i-1]\}.$$

Note that

$$I_i \wedge (i = n) \Rightarrow \varphi.$$

So the condition C is $i \neq n$.
Hence the loop.

```
s = a[0];
i = 1;
while( i != n )
{
  if( s > a[i] ) s = a[i];
  i++;
}
```

Sorting

Let us define what is sorted array

$$\mathrm{SortedArray}\,(a) \equiv \forall i\ 0 \leq i < \mathrm{length}\,(a) - 1 \Rightarrow a[i] \leq a[i+1].$$

Sorting

Let us define what is sorted array

$$\mathrm{SortedArray}\,(a) \equiv \forall i\ 0 \leq i < \mathrm{length}\,(a) - 1 \Rightarrow a[i] \leq a[i+1].$$

$$\mathrm{Sorted}\,(a, s) \equiv \forall i\ 0 \leq i < s \Rightarrow a[i] \leq a[i+1]$$

Choose the invariant $\mathrm{Sorted}\,(a, i)$ for a parameter i.

```
i = 0;
while( i != n)
{
  S; /* Do something to restore invariant */
  i++;
}
```

Choose the invariant $\mathrm{Sorted}\,(a, i)$ for a parameter i.

```
i = 0;
while( i != n)
{
  S; /* Do something to restore invariant */
  i++;
}
```

S itself is a loop statement.

$$\mathrm{Sorted}'(a, r, k) \equiv \forall j\ 0 \leq j < r\ a[j] \leq a[j+1]) \vee j = k - 1$$

```
k = i+1;
while( k != 0 )
{
  if( a[k-1] > a[k] ) /* swap a[k-1] and a[k] */
  k--;
}
```